cepton_sdk Documentation

Cepton Technologies

Sep 12, 2019

CONTENTS

CHAPTER

ONE

OVERVIEW

If a method is undocumented, consult the C/C++ SDK documentation, since many methods in this library are just wrapper functions.

1.1 Timestamps

Unless otherwise marked, all timestamps are seconds since the Unix epoch (UTC). Note that this differs from the C/C++ interface which uses microseconds.

CHAPTER

TWO

ERRORS

class cepton_sdk.**C_ErrorCode**
An enumeration.

class cepton_sdk.**C_Error**(*code=<C_ErrorCode.CEPTON_SUCCESS: 0>*, *msg=None*, *data=None*)
Error thrown by most sdk functions.

code
cepton_sdk.C_ErrorCode

class cepton_sdk.**C_Warning**

CHAPTER

THREE

SETUP

3.1 Types

class cepton_sdk.**ControlFlag**

An enumeration.

DISABLE_DISTANCE_CLIP = 8

DISABLE_IMAGE_CLIP = 4

DISABLE_NETWORK = 2

ENABLE_CROSSTALK_FILTER = 128

ENABLE_MULTIPLE_RETURNS = 16

ENABLE_STRAY_FILTER = 32

HOST_TIMESTAMPS = 64

3.2 Methods

cepton_sdk.**initialize**(*capture_path=None*, *capture_seek=0*, *control_flags=0*, *error_callback=None*, *port=None*, ***kwargs*)

Initializes SDK. Optionally starts capture replay.

Parameters **control_flags** – *cepton_sdk.ControlFlag*

cepton_sdk.**deinitialize**()

CHAPTER

FOUR

GENERAL

API for code that is agnostic to live/replay mode.

`cepton_sdk.`**`get_time`**`()`
: Returns capture replay time or live time.

`cepton_sdk.`**`get_timestamp`**`()`
: Returns unix timestamp

`cepton_sdk.`**`is_end`**`()`
: Returns true if next call to *wait* will throw *CEPTON_ERROR_EOF*

`cepton_sdk.`**`is_live`**`()`
: Returns true if capture replay is not open.

`cepton_sdk.`**`is_realtime`**`()`
: Returns true if live or capture replay is running.

`cepton_sdk.`**`wait`**`(`*duration=-1*`)`
: Resumes capture replay or sleeps for duration.

  If *duration* is *0*, then waits forever.

CHAPTER

FIVE

SENSORS

5.1 Types

class cepton_sdk.**SensorModel**
An enumeration.

FUSION_790 = 8

HR80M = 2

HR80T = 1

HR80T_R2 = 6

HR80W = 3

SORA_200 = 4

SORA_P60 = 11

VISTA_860 = 5

VISTA_860_GEN2 = 7

VISTA_M = 9

VISTA_P60 = 12

VISTA_X = 10

VISTA_X15 = 13

class cepton_sdk.**SensorInformation**

handle

serial_number

model_name

model

Type *cepton_sdk.SensorModel*

firmware_version

formal_firmware_version

last_reported_temperature

last_reported_humidity

last_reported_age

last_reported_hv

last_reported_optic_temperature

gps_ts_year

gps_ts_mont

gps_ts_day

gps_ts_hour

gps_ts_min

gps_ts_sec

return_count

is_mocked

is_pps_connected

is_nmea_connected

is_calibrated

is_over_heated

class cepton_sdk.**Sensor**(*sensor_info*)

information

Type *cepton_sdk.SensorInformation*

classmethod create(*serial_number*)

classmethod create_by_handle(*sensor_handle*)

classmethod create_by_index(*sensor_index*)

property handle

property serial_number

update()

Update sensor information.

Should be called often, to pull latest sensor information.

5.2 Methods

cepton_sdk.**has_sensor**(*sensor_serial_number*)

cepton_sdk.**get_sensors**(*cls=<class 'cepton_sdk.api.Sensor'>*)

Returns attached sensors.

Returns Dictionary of sensors, indexed by serial number.

CHAPTER

SIX

POINTS

6.1 Types

class cepton_sdk.**Points**(*n=0*)

3D points array.

timestamps_usec

timestamps

image_positions

distances

positions

intensities

return_strongest

return_farthest

valid

saturated

cepton_sdk.**combine_points**(*points_list*)

Combine list of points (*ImagePoints*, *Points*, etc).

List must be nonempty. :returns: combined_points

All point array classes support numpy indexing and assignment as if they were 1-d arrays:

```
n_points = len(points_1)
points_2[10:20] = points_1[:10]
```

Multiple point arrays can also be combined:

```
points = cepton_sdk.combine_points([points_1, points_2])
```

6.2 Methods

See *Listen*.

The following methods return points directly from the C SDK callback.

cepton_sdk.**listen_frames**(*callback*)
Register frames callback.

Throws error if *callback_id* is currently registered.

Returns callback_id

cepton_sdk.**unlisten_frames**(*callback_id*)
Unregisters frames callback.

Throws error if *callback_id* is not currently registered.

There are also listener classes that seamlessly handle accumulation and waiting.

class cepton_sdk.**FramesListener**
Listener for getting all sensor frames.

class cepton_sdk.**SensorFramesListener**(*serial_number*)

6.3 Export

cepton_sdk.export.**save_points_las**(*points*, *path*)
Save points to LAS file.

cepton_sdk.export.**load_points_las**(*load_path*, *cls=<class 'cepton_sdk.point.Points'>*)
Load points from LAS file.

Returns Points, extra_data

cepton_sdk.export.**save_points_ply**(*points*, *path*)
Save points to PLY file.

cepton_sdk.export.**save_points_pcd**(*points*, *path*)
Save points to PCD file.

CHAPTER

SEVEN

SERIAL

7.1 Methods

The following methods return serial lines directly from the C SDK callback.

cepton_sdk.**listen_serial_lines**(*callback*)

cepton_sdk.**unlisten_serial_lines**(*callback_id*)

There is also a listener class that seamlessly handle accumulation.

class cepton_sdk.**SerialLinesListener**

CHAPTER

EIGHT

CAPTURE REPLAY

To open/close capture files, use *cepton_sdk.initialize* and *cepton_sdk.deinitialize* methods respectively. The high level API methods will automatically resume the capture replay as necessary.

cepton_sdk.**open_replay**(*capture_path*, *capture_seek=0*, *enable_loop=False*)

cepton_sdk.**close_replay**()

cepton_sdk.capture_replay.**get_filename**()

cepton_sdk.capture_replay.**get_length**()

cepton_sdk.capture_replay.**get_position**()

cepton_sdk.capture_replay.**get_start_time**()

cepton_sdk.capture_replay.**get_time**()

cepton_sdk.capture_replay.**is_end**()

cepton_sdk.capture_replay.**is_open**()

cepton_sdk.capture_replay.**seek**(*t*)

cepton_sdk.capture_replay.**seek_relative**(*t*)

CHAPTER

NINE

EXPORT

Methods to import/export points to common file formats.

class cepton_sdk.export.**PointsFileType**
An enumeration.

CSV = 1

LAS = 2

PCD = 3

PLY = 4

cepton_sdk.export.**save_points**(*points*, *path*, *file_type=<PointsFileType.LAS: 2>*)
Save points to file.

Sets file extension based on type.

cepton_sdk.export.**load_points**(*path*, *file_type=None*)
Load points from file.

File type is inferred from extension.

Returns Points, extra_data

CHAPTER

TEN

SAMPLES

10.1 Multiple Sensors

Listing 1: samples/multiple_sensors.py

```
#!/usr/bin/env python3
"""
Sample script for getting points from multiple sensors simultaneously.
"""

import pprint

import cepton_sdk
import cepton_sdk.plot
from common import *

if __name__ == "__main__":
    # Variables
    capture_path = get_sample_capture_path()

    # Initialize
    cepton_sdk.initialize(capture_path=capture_path)

    # Get sensors
    sensors_dict = cepton_sdk.get_sensors()

    # Get points
    listener = cepton_sdk.FramesListener()
    points_dict = listener.get_points()
    del listener
    points_list = next(iter(points_dict.values()))
    points = points_list[0]

    # Plot
    cepton_sdk.plot.plot_points(points)
```

10.2 Single Sensor

Listing 2: samples/single_sensor.py

```
#!/usr/bin/env python3
"""
Sample script for getting points from a single sensor.
"""

import pprint

import numpy

import cepton_sdk
import cepton_sdk.plot
from common import *

if __name__ == "__main__":
    # Variables
    capture_path = get_sample_capture_path()

    # Initialize
    cepton_sdk.initialize(capture_path=capture_path)

    # Get sensor
    sensor = cepton_sdk.Sensor.create_by_index(0)
    pprint.pprint(sensor.information.to_dict())

    # Get points
    listener = cepton_sdk.SensorFramesListener(sensor.serial_number)
    points_list = listener.get_points()
    del listener
    points = points_list[0]

    # Plot
    cepton_sdk.plot.plot_points(points)
```

10.3 Advanced

10.3.1 Listen

Listing 3: samples/advanced/listen.py

```
#!/usr/bin/env python3
"""
Sample script for the different methods of getting points.
"""

import numpy

import cepton_sdk
from common import *


def on_frame(serial_number, points):
    print("Received {} points from sensor {}".format(
```

(continues on next page)

(continued from previous page)

```
        len(points), serial_number))


if __name__ == "__main__":
    # Initialize
    cepton_sdk.initialize(capture_path=get_sample_capture_path())
    sensors_dict = cepton_sdk.get_sensors()
    sensor = next(iter(sensors_dict.values()))

    callback_id = cepton_sdk.listen_frames(on_frame)
    cepton_sdk.wait(0.1)
    cepton_sdk.unlisten_frames(callback_id)

    # Get next frames for all sensors. Wait until data is available.
    listener = cepton_sdk.FramesListener()
    points_dict = listener.get_points()
    del listener

    # Get next frames for single sensor. Wait until data is available.
    listener = cepton_sdk.SensorFramesListener(sensor.serial_number)
    points_list = listener.get_points()
    del listener

    # Get large chunk of data
    listener = cepton_sdk.FramesListener()
    cepton_sdk.wait(10)
    points_dict = listener.get_points()
    del listener
    points = cepton_sdk.combine_points(points_dict[sensor.serial_number])
    print("Received {} seconds of data from sensor {}".format(
        numpy.ptp(points.timestamps), sensor.serial_number))
```

INDEX

L

M

O

P

R

S

T

U

V

W